

PR #51453 完整报告

vllm-project/vllm

[Performance] Register Triton W4A16 GEMM as a custom op

合并时间: 2026-09-02 11:55

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51453>

执行摘要

- 一句话: 将 Triton W4A16 GEMM 注册为自定义算子, 修复 compile mode 3 的 decode 吞吐回归
- 推荐动作: 值得精读, 尤其是维护 Triton 内核、自定义算子与 torch.compile 边界的工程师: 改动只有 60 余行, 但完整展示了「为什么要给内核划一条不透明边界」的推理链 (Python tile 选择 → 编译期常量折叠 → prefill 配置污染 decode)。建议重点关注三个设计点: (1) 用 direct_register_custom_op + fake impl 而不是 torch.compiler.disable/allow_in_graph 的取舍; (2) 门面函数保持签名不变以把调用方改动降为零; (3) 用 make_fx 断言图节点存在, 把「算子必须留在图里」变成可回归的契约。同时应在阅读时记住两个未闭环项: split-K 缺失导致的多数 shape 利用率不足, 以及端到端编译态性能仍无 CI 覆盖。

功能与动机

Issue #49699 「[Performance]: Compile mode 3 degrades triton w4a16 kernel performance in few request scenarios」报告: 以 Qwen 27B GPTQ-4bit 服务时, 把 compile mode 从 0 调到 3, decode 吞吐从 7.0 降到 4.8, 而 exllama 内核从 13 升到 26; 报告者临时通过提高 exllama 优先级绕过回归, 并担心「other triton kernels may also be interfered」。PR body 明确根因: 「Without this boundary, torch.compile traces the Python tile-selection logic and freezes the prefill configuration for decode workloads」, 因此需要一个自定义算子边界, 让 tile selection 使用真实运行时 shape, 参照 rdna_hybrid_w4a16.py 的做法。

实现拆解

本次变更集中在 `vllm/model_executor/kernels/linear/mixed_precision/triton_w4a16.py` 与 `tests/kernels/quantization/test_triton_w4a16.py`, 属于「内核边界 + 测试护栏」型改造。

1. 把真实实现改为内部函数: 原 `triton_w4a16_gemm` 重命名为 `_triton_w4a16_gemm_impl`, 函数体 (contiguous 断言、 $M, K = a.shape$ 、 $N = b_q.shape[1] * 8$ 、tile 选择、`triton_w4a16_gemm_kernel[grid]` 调用) 完全保留。改名是让 torch.compile 无法按函数名直接内联这段 Python 逻辑, 从而把 BLOCK_M/BLOCK_N/BLOCK_K 的选择 (含 if `group_size < BLOCK_K: BLOCK_K = group_size` 这一联动) 推迟到运行时, 而不是在编译期被冻结为 prefill 的 shape 配置。

2. 新增 fake 实现: `_triton_w4a16_gemm_fake` 只返回 `torch.empty((a.size(0), b_q.size(1) * 8), dtype=a.dtype, device=a.device)`, 即输出形状由 M 与打包权重第 1 维 (N//8) 乘 8 推导, 不执行任何计算、也不做形状断言。这一步是 `torch.compile` 能在 FakeTensor 下完成 tracing 与内存规划的前提。
3. 注册算子并保留同名门面: 通过新导入的 `direct_register_custom_op` 注册 `op_name="triton_w4a16_gemm"`, `mutates_args=[]` 表明不原地改写输入; 随后重新定义 `triton_w4a16_gemm(...)` 转发到 `torch.ops.vllm.triton_w4a16_gemm(...)`。门面函数签名与原函数逐参一致, `TritonW4A16LinearKernel` 及其上游调用方零改动, 降低回归面。
4. 测试配套: 新增 `test_triton_w4a16_gemm_fake_impl` (不需要 ROCm 硬件, 模块级 skip 已经排除非 ROCm 环境), 在 `FakeTensorMode()` 下断言输出 `shape/dtype/device`, 再用 `make_fx` 导图并断言图中存在 `call_function` 节点 `torch.ops.vllm.triton_w4a16_gemm.default`。这是防止后续有人把算子重新「降级」为普通 Python 函数的回归护栏, 也是整份改动里唯一可自动化的正确性证据。

配置、schema 与部署侧无改动: 不新增环境变量、不调整 `MPLinearKernel` 选择逻辑, 注册行为是模块导入时的副作用。验证侧数据来自 PR 描述: [pytest tests/kernels/quantization/{test_triton_w4a16,test_w4a16_kernel_selection,test_rocm_compressed_tensors_w4a16}.py](#) 共 41 个用例通过; GSM8K 由 50 样本扩到全量 1319 样本后精度不变; MI300 decode 吞吐先报 +20.8%, 复测 +18.4%。

关键文件:

- `vllm/model_executor/kernels/linear/mixed_precision/triton_w4a16.py` (模块 线性内核; 类别 source; 类型 data-contract; 符号 `triton_w4a16_gemm`, `_triton_w4a16_gemm_impl`, `_triton_w4a16_gemm_fake`): 唯一源码改动点: 把 Triton W4A16 GEMM 内部的 Python 实现重命名为 `_triton_w4a16_gemm_impl`, 新增 fake impl 与 `direct_register_custom_op` 注册, 并保留同名门面函数转发到 `torch.ops.vllm`, 是整个性能修复的落点。
- `tests/kernels/quantization/test_triton_w4a16.py` (模块 量化内核测试; 类别 test; 类型 test-coverage; 符号 `test_triton_w4a16_gemm_fake_impl`): 新增 `test_triton_w4a16_gemm_fake_impl`: 在 `FakeTensorMode` 下校验 fake 输出元信息, 并用 `make_fx` 断言图中出现 `torch.ops.vllm.triton_w4a16_gemm.default` 节点, 防止算子边界被无意改回普通函数。

关键符号: `triton_w4a16_gemm`, `_triton_w4a16_gemm_impl`, `_triton_w4a16_gemm_fake`, `test_triton_w4a16_gemm_fake_impl`

关键源码片段

`vllm/model_executor/kernels/linear/mixed_precision/triton_w4a16.py`

唯一源码改动点: 把 Triton W4A16 GEMM 内部的 Python 实现重命名为 `_triton_w4a16_gemm_impl`, 新增 fake impl 与 `direct_register_custom_op` 注册, 并保留同名门面函数转发到 `torch.ops.vllm`, 是整个性能修复的落点。

```
# 与 rdna_hybrid_w4a16.py 相同的做法: 把 Triton 内核包成不透明自定义算子,
# 让 torch.compile 只看到一个 call_function 节点, 不再内联并冻结 tile 选择逻辑。
```

```

from vllm.utils.torch_utils import direct_register_custom_op

def _triton_w4a16_gemm_fake(
    a: torch.Tensor,
    b_q: torch.Tensor,
    scales: torch.Tensor,
    qzeros: torch.Tensor | None,
    group_size: int,
    zp_bias: int = 8,
) -> torch.Tensor:
    # fake impl 只声明元信息：输出 N 维由打包权重第 1 维 (N//8) 乘 8 还原，
    # dtype 与 device 继承激活张量 a，不执行任何真实计算，也不做形状断言。
    return torch.empty((a.size(0), b_q.size(1) * 8), dtype=a.dtype, device=a.device)

direct_register_custom_op(
    op_name="triton_w4a16_gemm", # 对外暴露为 torch.ops.vllm.triton_w4a16_gemm
    op_func=_triton_w4a16_gemm_impl, # 真实 Triton 实现，改名后不再被编译期内联
    mutates_args=[], # 不原地写入输入，输出为新张量，对 CUDA Graph 捕获友好
    fake_impl=_triton_w4a16_gemm_fake, # 让 FakeTensor / torch.compile 能完成 tracing
)

def triton_w4a16_gemm(
    a: torch.Tensor,
    b_q: torch.Tensor,
    scales: torch.Tensor,
    qzeros: torch.Tensor | None,
    group_size: int,
    zp_bias: int = 8,
) -> torch.Tensor:
    # 保留原函数名作为门面 (facade)，TritonW4A16LinearKernel 等调用方无需改动。
    # 真正的 tile 选择发生在 _triton_w4a16_gemm_impl 内部，按运行时真实 M 计算，
    # 因此 decode 的小 batch 不会再被 prefill 时期冻结的 BLOCK_M/BLOCK_N/BLOCK_K 拖累。
    return torch.ops.vllm.triton_w4a16_gemm(a, b_q, scales, qzeros, group_size, zp_bias)

```

tests/kernels/quantization/test_triton_w4a16.py

新增 test_triton_w4a16_gemm_fake_impl：在 FakeTensorMode 下校验 fake 输出元信息，并用 make_fx 断言图中出现 torch.ops.vllm.triton_w4a16_gemm.default 节点，防止算子边界被无意改回普通函数。

```

from torch._subclasses.fake_tensor import FakeTensorMode
from torch.fx.experimental.proxy_tensor import make_fx

def test_triton_w4a16_gemm_fake_impl():
    # FakeTensorMode 下不执行真实 Triton 内核，只校验 fake impl 的元信息与图结构，
    # 因此该用例可在无 GPU 计算的前提下守住算子边界。

```

```

with FakeTensorMode():
    a = torch.empty((4, 256), device=device, dtype=torch.float16) # [M, K]
    b_q = torch.empty((256, 64), device=device, dtype=torch.int32) # [K, N//8]
    scales = torch.empty((2, 512), device=device, dtype=torch.float16) # [K//G, N]

    out = triton_w4a16_gemm(
        a=a,
        b_q=b_q,
        scales=scales,
        qzeros=None, # 对称 uint4b8, 走 zp_bias=8 分支
        group_size=128,
    )

    # N = 64 * 8 = 512, dtype / device 必须与激活保持一致
    assert out.shape == (4, 512)
    assert out.dtype == a.dtype
    assert out.device == a.device

    # 关键回归护栏：导出后的图里必须出现 vLLM 自定义算子节点，
    # 而不是被打散成 Python / Triton 逻辑，否则 tile 选择又会退回编译期折叠。
    graph = make_fx(
        lambda a, b_q, scales: triton_w4a16_gemm(
            a, b_q, scales, qzeros=None, group_size=128
        )
    )(a, b_q, scales)

    assert any(
        node.op == "call_function"
        and node.target == torch.ops.vllm.triton_w4a16_gemm.default
        for node in graph.graph.nodes
    )

```

评论区精华

review 讨论量不大但有两处值得记录：

1. 验证口径之争 (testing)：作者最初只跑 50 样本 GSM8K，@tjtanaa 明确要求扩到全量：「can you rerun with gsm8k full dataset rather than just 50 samples. Because 50 samples has high variance」。作者随后用 1319 样本复测并给出「no accuracy change, decode 吞吐约 +18.4%」，吞吐收益从 +20.8% 收敛到 +18.4%，说明小样本口径确实会放大改善幅度。
2. 遗留的性能诉求 (performance, 未解决)：@hnhzyz 在合并前提出「Could you please add split-k support to the triton kernel? The GPU is underutilized for most shapes」，本 PR 未处理，split-K 缺失仍是该内核在多数 shape（尤其小 M decode）下利用率不足的已知短板。
3. 流程与结论：@hongxiayang 在 issue 中 cc @tjtanaa 请求 review；claude[bot] 因 PR 来自 fork 而关闭自动 review；@tjtanaa 两次 APPROVED（「LGTM」「LGTM. Thanks

for addressing the issue.」），@AndreasKaratzas 触发 CI 并完成合并。

- GSM8K 精度验证的样本量口径 (testing): 作者以全量 test set 复测得「no accuracy change」，吞吐收益从 +20.8% 修正为约 +18.4%，验证口径被收紧后结论仍然成立。
- Triton W4A16 内核缺少 split-K 支持 (performance): 本 PR 未处理，属于后续独立优化项；当前内核在小 M decode 场景的利用率短板仍然存在。
- 以自定义算子边界隔离 torch.compile 的设计选择 (design): reviewer 未提出异议，@tjtanaa 两次 LGTM 并「Thanks for addressing the issue」，方案按原样合并。
- fork PR 的自动 review 被禁用 (other): 本 PR 实际依赖人工 reviewer (@tjtanaa、@AndreasKaratzas) 把关，无自动化深度审查记录。

风险与影响

• 风险:

1. 量化主路径回归面：所有在 ROCm/CUDA 上命中 TritonW4A16LinearKernel (compressed-tensors W4A16、GPTQ uint4b8/uint4) 的请求都要经过 torch.ops.vllm 调度器。门面函数签名保持一致，正常情况下调用方无感；但任何直接 import 内部实现、monkeypatch 原函数或依赖 triton_w4a16_gemm 为纯 Python 函数的代码（含第三方插件）会失效——重命名 triton_w4a16_gemm 为 _triton_w4a16_gemm_impl 属于隐式接口变更。
2. compute 与 CPU 开销的取舍：算子变为不透明后，_triton_w4a16_gemm_impl 仍是 Python 函数，每次调用都要在 Python 层重算 tile 启发式与执行 assert a.is_contiguous() 等断言。本次测量显示净收益为正，但收益只在 MI300/gfx90a 级别硬件、compile mode 3 场景下被验证，收益是否在其它 ROCm 卡型与 CUDA 路径 (can_implement 同时允许 is_cuda()) 成立缺少数据。
3. fake impl 与真实实现的契约漂移：fake impl 只按 (a.size(0), b_q.size(1) * 8) 推断输出，不校验 K 与 qzeros 形状。一旦后续改动输出形状或 dtype（例如引入 split-K 中间缓冲、换输出 dtype），fake 未同步更新会造成编译期静默错配，这类错误只在图执行时才暴露。
4. 验证覆盖不足：新增测试只覆盖 fake impl 与图节点存在性，端到端「compile mode 3 下吞吐与精度」依赖人工跑 GSM8K，未纳入 CI；test_triton_w4a16_gemm_fake_impl 里固定 M=4、group_size=128，对更多 shape 组合没有约束。
5. 导入期副作用：算子注册发生在模块导入时，多次导入 / 子进程重复注册的幂等性由 direct_register_custom_op 保证，本材料未体现其实现细节，属于低概率但需留意的点。
 - 影响：对用户：ROCm MI300 及更新硬件上以编译模式（尤其 compile mode 3 + FULL_AND_PIECEWISE CUDA Graph）服务 W4A16/GPTQ-4bit 模型的用户直接受益，decode 吞吐 +18%~21% 且精度不变，不再需要用「手动调高 exllama 内核优先级」这种绕过手段来规避回归。对系统：这条回归的本质是「Python 启发式逻辑被 torch.compile 提前折叠」，本 PR 把该问题在 ROCm Triton W4A16 路径上关闭，同时示范了一个通用解法 (custom op + fake impl + 门面函数)，可复制到其它仍被编译期冻结的 Triton 内核。对团队：AMD/ROCm 内核维护者获得一个新的隐性维护契约——fake impl 必须与真实实现的输出元信息保持同步；同时 @hnhzyz 提出的 split-K 请求把「内核实现效率」这一议题留给后续 PR。
 - 风险标记：量化主路径变更，

torch.compile 边界变更，隐式接口重命名，split-K 缺失（小 shape 利用率低），端到端编译态性能未进 CI

关联脉络

- PR #51692 [ROCm][Perf] Add breshuffled blockscaled fp8 GEMM: 同属 vllm/model_executor/kernels/linear/ 下的 ROCm 量化 GEMM 内核工作，同样涉及内核注册与选择体系的扩展，可对照本 PR 观察 ROCm 线性内核层的演进节奏。
- PR #55465 [Fast Start] Support fp4: 同时改动 vllm/model_executor/kernels/linear/nvfp4/*（含 trtllm/flashinfer/cutlass 多后端注册）与量化内核入口，与本 PR 一样在改动量化内核的对外边界与注册方式，属于同类「内核可插拔化」脉络。
- PR #56145 [Core] MRV2 support for fast-prefill: 同样围绕 torch.compile / 编译态下的执行配置（fast-prefill 与 V2 门禁）做调整，本 PR 则是编译态冻结 tile 配置的反向案例，两者共同反映编译边界与运行时 shape 的耦合问题。
- PR #54889 [DCP][Kernel][Perf] Fuse the empty-shard LSE mask into the A2A pack kernel: 同属以内核级手段消除每层额外 kernel 开销的 ROCm 性能优化，且关注 CUDA Graph / 编译态下的 kernel 数膨胀，与本 PR 的「减少编译期对内核的干预」目标同向。